Sistemas Informáticos

Curso 2004 - 2005

Agentes capaces de deducir de forma cooperativa relaciones causa efecto en un entorno estructurado

David González Gómez

Marcos Gómez Arranz

Raúl Vallejo Gómez

Dirigido por:

Matilde Santos Peñas

Dpto. Arquitectura de Computadores y Automática

Facultad de Informática

Universidad Complutense de Madrid

Resumen

En este proyecto se ha desarrollado una arquitectura de sistema multi-agente capaz de tomar decisiones coordinadas en conjunto.

A lo largo de esta memoria se explica cuales han sido las decisiones de diseño de los diversos puntos de los que consta la arquitectura. También y dado que la arquitectura no parte de cero, se comentan los diversos entornos de ejecución estudiados y por qué se ha escogido JADE como el entorno en el que van a correr los agentes.

Por último se mostrará un ejemplo de esta arquitectura, en el que a través de una implementación específica se prueba un ejemplo de la ejecución en un entorno estructurado.

Abstract

In this project, we have developed a multi-agent system architecture, which is able to take coordinated decisions by the whole group of agents.

Throughout this text there are explanations about the design decisions made in the diferent points that the architecture is based on. Also, due to the architecture is built over an agent environment, there are discussions about several running environments and why JADE has been chosen.

At last, an example of the architecture will be shown throught a specific implementation, and an example of the execution into a structured environment.

Autorización de uso

Nosotros, los integrantes del grupo desarrollador del proyecto “Agentes capaces de deducir de forma cooperativa relaciones causa efecto en un entorno estructurado” de la asignatura de Sistemas Informáticos: David González Gómez, Marcos Gómez Arranz y Raúl Vallejo Gómez, autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales, el trabajo presentado, siempre que seamos mencionados expresamente como los autores.

Firmas de los miembros del grupo:

David González Gómez Raúl Vallejo Gómez Marcos Gómez Arranz

Madrid, 30 de junio de 2005

Índice general

Resumen I

Abstract I

Autorización de uso II

Índice de figuras V

Capítulo 1. Introducción 1

1.1. Agentes. 1

1.2. Sistemas Multi-Agente. 1

1.3. Entornos de desarrollo. 2

1.3.1. ABLE (Agent Building and Learning Environment) *3*

1.3.2. JADE (Java Agent Development) *3*

1.3.3. JADE vs ABLE *4*

1.4. Arquitectura 4

1.4.1. Capas *5*

Capítulo 2. Entorno de desarrollo 7

2.1. Comparativa 8

2.2. Agentes en JADE 12

Capítulo 3. Arquitectura 13

3.1. Definiciones 13

3.2. Primera aproximación 13

3.3. Centralización vs Distribución 15

3.4. Agentes y Estados 16

3.4.1. Modelo 1 . . . . . *17*

3.4.2. Modelo 2 . . . . . *17*

3.4.3. Conclusiones *18*

3.5. Aprendizaje 18
3.6. Colaboración 19
3.6.1. Votación *20*
3.7. Boceto de especificación 20
3.7.1. Definiciones de especificación *21*
3.7.2. Especificación de bajo nivel *22*
3.7.3. Especificación de alto nivel *22*
3.7.4. Especificación del comportamiento completo *23*
3.7.5. Niveles de comunicación *23*
3.8. Ejemplo del nido 25
3.9. Entrada del sistema 26
3.9.1. Definición del problema *26*
3.9.2. Estrategias *26*
3.9.3. Estados de un agente *26*

Capítulo 4. Especificación 27
4.1. Conceptos 27
4.2. Especificación 28
4.3. Especificaciónn de un Problema 29
4.4. Especificación de un Agente 29

Capítulo 5. Ejemplo implementado 31
5.1. Interfaz gráfica de entrada 32
5.2. Entorno de simulación 35
5.3. Sistema de agentes 39

Bibliografía 42

Bibliografía 43

Apéndice A. Fórmulas 44

Apéndice B. Robots 48

Índice de figuras

2.1.	Esquema de la arquitectura distribuida de una plataforma en JADE.	8
2.2.	Escalabilidad en comunicaciones en un mismo contenedor.	9
2.3.	Escalabilidad en comunicaciones en distintos contenedores pero en la misma plataforma.	10
2.4.	Escalabilidad en comunicaciones en distintas plataformas frente a la misma plataforma en distintos contenedores.	11
3.1.	Arquitectura inicial del sistema.	14
3.2.	Modelo 1.	17
3.3.	Modelo 2.	18
3.4.	Niveles de actuación.	24
4.1.	Sistema a alto nivel.	27
4.2.	Sistema a bajo nivel.	28
4.3.	Niveles de percepción.	28
4.4.	Esquema de la arquitectura general	30
5.1.	Edición del mapa del entorno.	33
5.2.	Diseño de los estados.	34
5.3.	Valores iniciales de los elementos.	34
5.4.	Captura de la aplicación en ejecución.	35
5.5.	Primera parte del diagrama de clases de Entorno	37
5.6.	Segunda parte del diagrama de clases de Entorno	38
5.7.	Diagrama de clases de la arquitectura implementada	40

A.1. Diagrama de clases de *Formula* 47

CAPÍTULO 1

Introducción

1.1. Agentes.

La definición de un agente no es una tarea fácil. Intuitivamente se denomina agente a una entidad capaz de interactuar con su entorno. Con esta generalidad aparece una amplia diversidad de formas, un gran abanico de clasificaciones y, en resumen, una linea de investigación muy interesante.

En este proyecto se han usado agentes software de la forma más general que se ha podido y que cumpla con las restricciones impuestas: capacidad de comunicación y coordinación con otros agentes y capacidad de tener en cuenta propiedades de el entorno en el que se encuentra.

1.2. Sistemas Multi-Agente.

Un sistema multi-agente trata de agrupar las tareas que puede realizar cada uno de los agentes independientes para que en conjunto se realice algo al menos igual que la suma de cada uno por separado, ya sea simplificando la labor de desarrollo a través de una modularización del proceso, o bien de una forma más interna con las relaciones entre los propios agentes. Típicamente un sistema multi-agente es capaz de realizar tareas en las que cada agente por separado no tiene una información completa del problema o capacidad para realizarla.

Por un lado un sistema multi-agente se puede ver como un conjunto de agentes entre los que no se conoce nada más que un lenguaje de comunicación, con el que conversan para llevar

a cabo la tarea requerida de cada uno. Para que esto sea posible, en algún lugar deben estar anotadas las tareas que puede realizar cada agente, y una forma de mandarle la información que es requerida para realizar esa tarea. De este modo aparecen los lenguajes de comunicación entre agentes y un "elemento" común conocido por todos con el cual obtienen información de los demás agentes.

Por otro lado, se puede ver como un proceso más interno entre los diversos agentes. Una constancia de que puede haber más agentes igual a uno dado, y por lo tanto un conocimiento superior de las capacidades del sistema completo. Esto se puede explotar para mejorar la eficiencia con la que se realiza una tarea, a través de la coordinación. Si a esto se le añade la capacidad de conocer propiedades del entorno, se obtiene un sistema capaz de deducir relaciones causa efecto, haciendo uso o bien de la prueba y error hasta optimizar la tarea o bien gracias a la experiencia ganada puntualmente por todos los individuos. Como veremos, la segunda es la que más problemas conlleva.

1.3. Entornos de desarrollo.

El desarrollo y ejecución del sistema se ha realizado bajo un entorno ya creado. Hay múltiples marcos de trabajo o *Frameworks* para realizar esta tarea. En un principio se barajó la posibilidad de utilizar el entorno desarrollado por unos compañeros como proyecto de Sistemas Informaticos del año 2004 [**Mestre04**]. En él, se desarrolla una implementación de la arquitectura IMPACT (Interactive Maryland Platform for Agents Collaborating Together) en la que se encuentran diversos elementos para la creación de un sistema multi-agente heterogéneo. Cabe destacar la capacidad para comunicación transparente a través de unservicio de páginas amarillas, en el que se registran tanto los agentes como sus funciones disponibles. También se usa este servicio para poner en comunicación a los agentes que hagan la petición de un servicio con el que tenga esa función requerida.

IMPACT es una arquitectura muy flexible que permite una heterogeneidad abrumadora para un sistema multi-agente [**Subrahmanian00**]. Sin embargo en este proyecto no se requiere tanta diversidad debido a que la gran mayoría de los agentes van a ser iguales, haciendo tareas

conocidas por todos y por lo tanto con poca necesidad de publicar y hacer trabajo requerido desde otros agentes similares en rango. Por ello se decidió hacer una arquitectura mucho más simple que IMPACT pero que tuviese en cuenta esta heterogeneidad limitada, incrementando así el rendimiento.

Se barajaron otros entornos de desarrollo y ejecución que están a un nivel de madurez mayor gracias a que llevan años en entornos de producción o en investigación. Destacar algunos de ellos:

1.3.1. ABLE (Agent Building and Learning Environment).

Está desarrollado por IBM [**Bigus01**], y no está orientado inicialmente a crear sistemas multi-agente. Permite crear agentes en forma de *AbleBeans* o *AbleAgents*, que son clases en JAVA desde las que, utilizando herencia, se desarrollan los agentes necesarios. Haciendo uso de un editor de agentes se obtiene una visión gráfica del conjunto completo de la aplicación. Otra de las ventajas de este entorno es la gran facilidad para incorporar diversos mecanismos de inteligencia artificial a los propios agentes, tales como redes neuronales, árboles de decisión, Bayes...

1.3.2. JADE (Java Agent Development).

Su desarrollo actual se lleva a cabo en Telecom Italia Lab, aunque gracias a su licencia libre (Lesser General Public License) permite que realmente sea desarrollado por una comunidad global muy activa. Esto le da otra de sus grandes ventajas, como es la documentación generada por toda esa comunidad y una realimentación continua. Es un entorno probado y con múltiples proyectos en producción. Está orientado a la creación de sistemas multi-agente desde una perspectiva interna al agente. Se distingue entre el agente en sí y sus comportamientos, que son intercambiables y, lógicamente, es posible basarse en cualquiera de ellos para desarrollar un comportamiento nuevo. JADE cumple las especificaciones de la FIPA (Foundation for Intelligent Physical Agents), encargada de desarrollar estándares para la interoperabilidad entre plataformas de agentes software.

1.3.3. JADE vs ABLE.

Entre estos dos entornos se escogió JADE principalmente por su mayor semejanza con la base de la arquitectura que se ha diseñado. Proporciona facilidades como el desarrollo extremadamente rápido de agentes dentro de un sistema multi-agente, desde una perspectiva de la interoperabilidad total tanto entre plataformas que cumplan con FIPA, como entre los propios agentes independientemente de la ubicación/dispositivo/distancia a la que se encuentren.

Able tiene problemas importantes en estos puntos, siendo por ejemplo bastante engorrosa la comunicación explícita entre un cliente y un servidor, como se pretende hacer con la arquitectura.

1.4. Arquitectura

El desarrollo del proyecto se centra en una arquitectura capaz de dar las facilidades para que un conjunto de agentes, algunos con funciones específicas y otros que implementan la inteligencia de los componentes físicos (robots por ejemplo) puedan realizar una tarea específica satisfactoriamente.

Estos agentes, a raíz de la arquitectura, tienen un conjunto de estados y también un conjunto de estrategias a elegir.

Los estados serán rangos del dominio en el que se define cada una de las variables que tiene un agente. Se agrupan en forma de **máquina de estados finita** y vienen definidos como entrada del sistema. Las transiciones entre los estados de esta máquina de estados finita vienen definidas de forma natural por el comportamiento de las variables del agente, y por factores externos que alteren los valores de estas variables. El objetivo de este agrupamiento de valores en estados es restringir las estrategias a los estados. Esta restricción será opcional dependiendo del problema.

De esta manera se abordan dos tipos de problemas: **resolución** y **optimización**. Esto da lugar a dos implementaciones muy distintas de un sistema. La primera se ajusta a problemas reales en los que hay parte de información desconocida en el momento inicial y es necesario que los agentes obtengan experiencia para tener un mayor conocimiento y por lo tanto aumentar su capacidad para realizar una tarea. La segunda implica la ejecución del problema un número posiblemente elevado de veces, con un conocimiento del problema completo desde un inicio, aunque este esté implicito.

1.4.1. Capas.

Dividir la arquitectura en capas da múltiples ventajas como se demuestra en una ingente cantidad de programas, protocolos y otros productos que se han desarrollado dentro del ámbito de la Informática a lo largo de su historia. La propia Informática está dividida en capas desde la física y lógica hasta la aplicación. Por ello parece natural que esta arquitectura siga con el mismo principio.

De la arquitectura con la que empezamos el proyecto a la final, las modificaciones han sido muy grandes y en algunos casos traumáticas. A lo largo del capítulo referido a la arquitectura se mostrarán estos cámbios y la arquitectura final con la que se ha implementado un pequeño ejemplo. Inicialmente había tres capas:

- Capa de gestión de agentes, encargada de la comunicación, la creación y destrucción de agentes haciendo uso de las facilidades que proporciona JADE, con el agente especial (AMS) y la gestión de servicios, también con el agente especial de JADE referido para este propósito (DF).
- Capa de agentes y control, en la que se realiza el aprendizaje y el control del sistema. Se comunica con la anterior con mensajes ACL y en ella están todos los agentes propiamente dichos: el conjunto homogéneo de agentes encargados de realizar la tarea, los agentes encargados de proporcionar información al sistema a través de tratar la información llegada desde los dispositivos físicos (una cámara o sensores de posición en general), y el resto de agentes necesarios para abordar el problema.

- Capa física, donde se encuentran los interfaces para comunicar con las pártes físicas del sistema a los agentes encargados de tratar la información en la capa anterior. Aquí se realiza la gestión de los sensores y/o actuadores, las rutinas de movimiento básico de los dispositivos físicos y la comunicación hasta el equipo o equipos donde esté ejecutandose el sistema.

Todos estos puntos son los que van a ser tratados en los siguientes capitulos de manera amplia, tal y como se han ido produciendo a lo largo de este curso.

CAPÍTULO 2

Entorno de desarrollo

El uso de JADE para el desarrollo del proyecto situa al resto de la arquitectura en un punto con una serie de facilidades que es necesario ver para obtener los cimientos sobre los que se asentará la siguiente capa.

El objetivo de JADE es simplificar el desarrollo de sistemas multi-agente asegurando el cumplimiento de los estándares a través de un conjunto de servicios y agentes que sigan las especificaciones de la FIPA: servicio de nombres y servicio de páginas amarillas, transporte de mensajes y protocolos de interacción.

También permite distribuir este servicio por varios *computadores* con lo que de manera transparente proporciona una capacidad de un sistema multi-agente distribuido por distintas zonas. Por lo tanto, se puede tener un único sistema de agentes realizando una tarea que requiera su distribución de componentes físicos. Evita así problemas con tener un único dispositivo de recepción de información conectado a un único computador por el cual se necesita realizar toda la comunicación.

Esta distribución se realiza haciendo uso de una máquina virtual de JAVA en cada máquina en la que se vaya a ejecutar un subconjunto de los agentes, y sobre ella una única aplicación de JADE en cada máquina. La arquitectura de comunicación proporcionada, mostrada en la figura 2.1 ofrece una mensajería flexible y eficiente, gestionando el propio JADE una cola de de mensajes ACL de entrada para cada agente. Así cada agente puede acceder a su cola de mensajes por una combinación de varios modos: bloqueante (*blocking*), sondeo (*polling*), tiempo límite (*timeout*) y encaje de patrones (*pattern matching*).

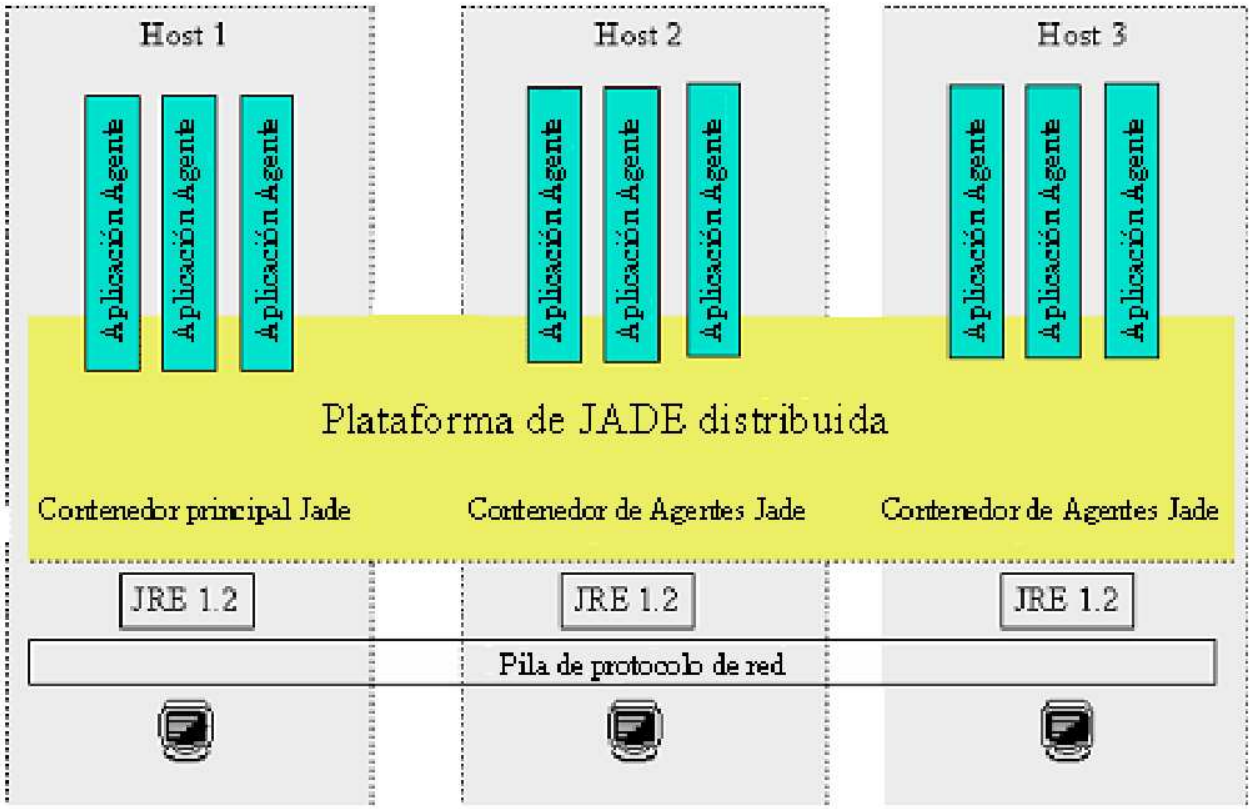

FIGURA 2.1. Esquema de la arquitectura distribuida de una plataforma en JADE.

La arquitectura de JADE permite también la transparencia de protocolos de comunicaciones entre estas máquinas. Si dos agentes en distintas máquinas deben comunicarse entre sí, es exactamente igual que si estuviesen en la misma máquina al nivel en el que se define la arquitectura aquí planteada, haciendo uso de Java RMI (*Remote Method Invocation*) o protocolos implementados en la versión de JADE utilizada: HTTP (*Hypertext Transfer Protocol*) e IIOP (*Internet Inter-Orb Protocol*). La elección del más idóneo en cada momento es cuestión de JADE.

2.1. Comparativa

En los gráficos de las figuras 2.2, 2.3 y 2.4 se muestra una comparativa en la que se estudia el comportamiento de JADE con respecto a las comunicaciones [**Cortese02**], que es en gran parte lo que incitó a utilizar esta plataforma para la base de la arquitectura. En el eje de ordenadas se

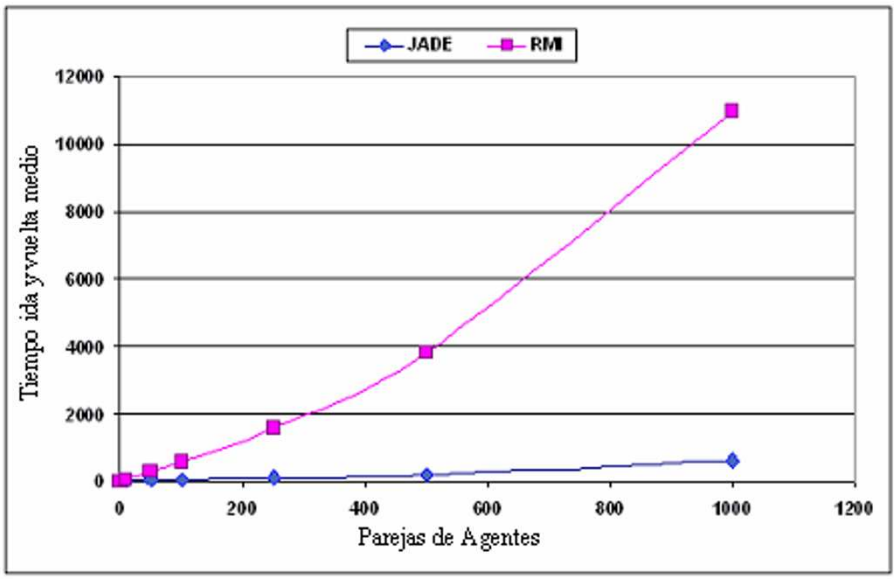

FIGURA 2.2. Escalabilidad en comunicaciones en un mismo contenedor.

muestra el tiempo medio de comunicación de ida y vuelta. En el eje de abcisas, el número de parejas en el sistema comunicándose.

La escalabilidad en comunicaciones sobre una misma máquina virtual de JAVA (JVM) se desarrolla de una manera exponencial tanto si se implementa usando diréctamente RMI como usando la implementacion de JADE, sin embargo comparativamente JADE tiene una constante multiplicativa muy inferior que en la práctica le da ventaja frente a la otra opción. Esto se debe principalmente a que tiene en cuenta la ubicación de los agentes que se comunican y si estos se encuentran bajo la misma máquina virtual permite que se llame directamente a los métodos en vez de usar mecanismos genéricos de invocación de métodos, eliminando así la sobrecarga que esto produce. Con esto se puede explotar la localidad de los grupos de agentes y pensar en diseñar un sistema en el que agentes que necesiten una comunicación muy frecuente se encuentren ubicados en el mismo computador.

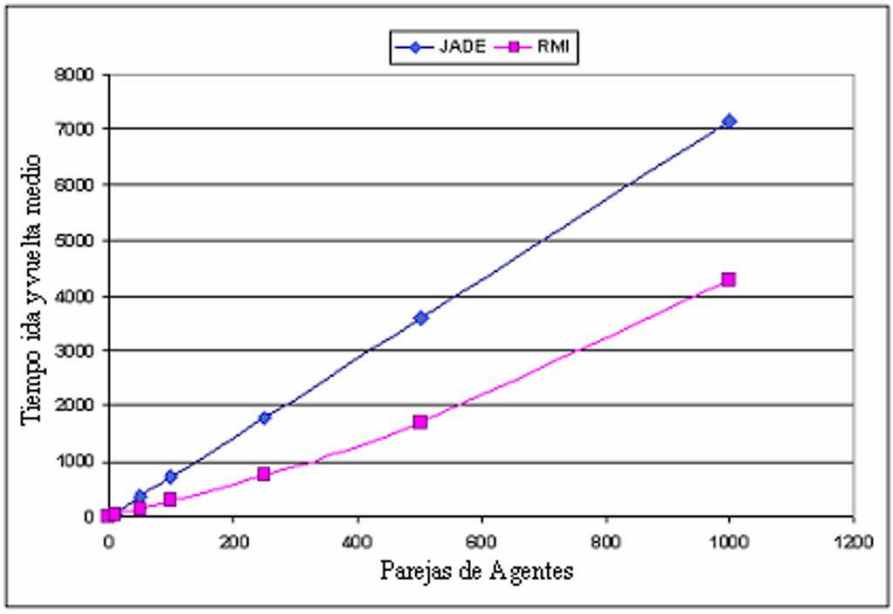

FIGURA 2.3. Escalabilidad en comunicaciones en distintos contenedores pero en la misma plataforma.

La escalabilidad en distintas JVM obliga a JADE a usar la implementación de RMI. Los agentes ubicados en distintos computadores producen un tiempo superior a una implementación explícita de RMI. Esto se debe al alto nivel de abstracción introducido por las capas de comunicación de JADE, destinadas a tener en cuenta el mecanismo de enrutado de los mensajes, la creación de los mensajes ACL y lógicamente, la sobrecarga que produce tener que elegir la mejor manera para hacer llegar ese mensaje. Con el uso explícito de RMI basta con mirar un registro de ubicación de agentes.

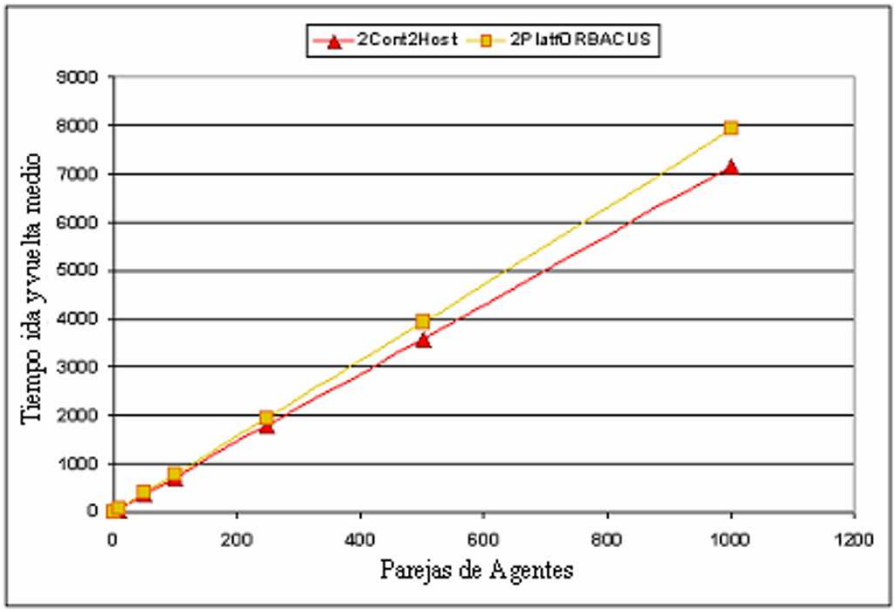

FIGURA 2.4. Escalabilidad en comunicaciones en distintas plataformas frente a la misma plataforma en distintos contenedores.

En distintas plataformas ubicadas en distintos computadores se da un rendimiento muy similar a la configuración de agentes de la misma plataforma, en distintos contenedores.

2.2. Agentes en JADE

La ejecución de cada agente se lleva a cabo con un hilo de proceso (*thread*) cada uno. Además de los recursos que da JAVA para crear estos hilos, JADE permite la planificación de comportamientos coordinados con estos hilos de una manera que no compromete los recursos del sistema.

Los comportamientos son el punto fuerte de JADE y en lo que se basa la arquitectura del proyecto en sus distintas fases. Más allá de un componente implementado en JADE, los comportamientos son, abstrayendo su concepto, la cualidad que permite que los agentes varíen su forma de actuar dinámicamente según un determinado estado y una situación exterior. De hecho, y como se verá en la sección 3.4 según cómo se utilice esta distinción entre comportamiento y estado, dará lugar a dos modelos de aprendizaje y coordinación distintos.

CAPÍTULO 3

Arquitectura

En este capítulo se describe la evolución de la arquitectura desarrollada en el proyecto, dando lugar a varias sub arquitecturas muy distintas las cuales aunque resolvían problemas concretos de una manera efectiva, tenían grandes lagunas en otros aspectos que se vieron según se profundizaba en la diversidad de problemas.

3.1. Definiciones

- **Estado del agente**: conjunto de valores de las variables (relevantes) del agente.
- **Estado del autómata**: Resume un rango de valores de estados del agente.
- **Estrategia/Comportamiento**: Lo que cada agente hace en cada momento, implicando también la forma en que se modifican las variables (lleva asociada un conjunto de fórmulas).
- **Acción individual**: aplicación de una estrategia por un agente en un determinado momento.
- **Acción conjunta**: el conjunto de las acciones individuales de todos los agentes en un determinado momento.

3.2. Primera aproximación

La arquitectura inicial ya ha sido presentada anteriormente en la introducción. La figura 3.1 muestra un esquema de su estructura.

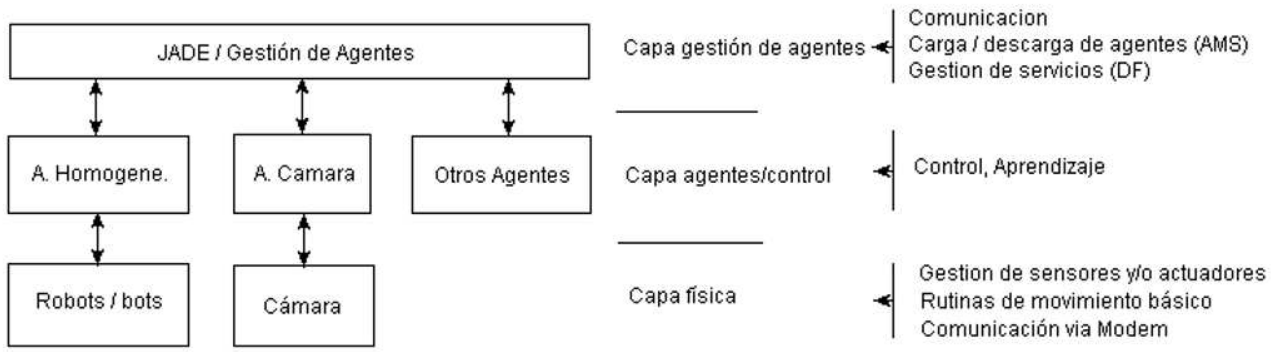

FIGURA 3.1. Arquitectura inicial del sistema.

La **capa física** inicialmente está compuesta por los robots (o sus simulaciones) y una cámara (o sensor de posición similar) que observa el entorno. Con respecto a los robots se ha decidido que deberían ser capaces de hacer una serie de movimientos por si solos, como por ejemplo seguir un pasillo hasta llegar a una esquina, o dirigirse hacia un emisor que detecte algún sensor.

En comunicación con esta capa encontramos la **capa de agentes**, que es la que se va a desarrollar en más profundidad en este proyecto. En esta capa se hará la toma de decisiones, se decidirán estrategias, y se aprenderá.
La capa se compone de un conjunto de agentes homogéneos, que serán los encargados de controlar los robots a un cierto nivel. También en esta capa habrá un agente encargado de manejar lo que la cámara recibe, para proporcionarlo a aquellos agentes que lo pidan de una forma legible. Una de las decisiones que debe tomarse en este punto es si se desea que las cosas se hagan de forma centralizada en algunos agentes, o distribuida entre los agentes homogéneos. Por ejemplo, las estrategias que sigue cada robot pueden ser decididas por un agente de forma centralizada, o cada agente puede escoger su estrategia (considerando la que sea mejor para el sistema en general).
Como “Otros Agentes” se entiende a los agentes que se encarguen de aquello que no distribuyamos entre los agentes homogéneos. Algunos agentes candidatos serían por ejemplo servidores de estrategias de grupo, de aprendizaje, de roles (estrategias individuales), de información del entorno, etcétera.

La **capa de gestión de agentes** es de la que se encarga JADE. Usando este sistema, el servicio de páginas amarillas (Directory Facilitator) y páginas blancas (Agent Management System) serían también agentes, por lo que pertenecerían a la capa de agentes.

3.3. Centralización vs Distribución

Los sistemas multi-agente vienen normalmente dotados de una arquitectura distribuida en cuanto a la toma de decisiones y el aprendizaje. Esto es, no hay un agente encargado sólo de aprender o sólo de decidir, sino que cada agente es capaz de hacer estas cosas. Estos comportamientos se describen por medio de la Teoría de Juegos.
Pero esto conlleva varios supuestos, como que los agentes deben tener un conocimiento común sobre el entorno, y cada agente ha de saber la posición y la estrategia que va a seguir cada uno de los demás agentes en cada momento.
Por el contrario, un acercamiento al problema mediante la centralización de algunos de los servicios haría de los agentes homogéneos simples terminales para la comunicación con los robots, desligándolos del concepto clásico de Agente Inteligente [**Vlassis03**].

Desde el punto de vista de la implementación tambien hay una gran diferencia entre estos antónimos. Una arquitectura centralizada implica la disposición de un conjunto de recursos limitados para gestionar todo el proceso y la comunicación desde un único punto. Si el sistema multi-agente contiene un número elevado de agentes que requieren de la decisión y aprendizaje puede originar cuellos de botella tanto en la comunicación por la red de interconexión, como en la propia ejecución de la aplicación en la máquina. Sin embargo un sistema distribuido evita estos problemas a costa de una complejidad superior en la toma de decisiones en conjunto y de una memoria de aprendizaje situada en los diversos nodos. Así se ve que la decisión de una arquitectura u otra es algo crítico para el futuro desarrollo de la arquitectura final. Por ello se tratará de profundizar todo lo posible en ella hasta que sea obligatorio escoger entre ambas, obteniendo así toda la información posible para llevar a cabo la decisión.

3.4. Agentes y Estados

Los agentes deben basar su comportamiento en una serie de estados, que vienen determinados por los valores de las propiedades internas de los mismos agentes. El número de estados no debe ser muy grande, pero debe ser ampliable.

Estos estados han de ser dinámicos, para lo que se ha desarrollado un conjunto de clases en java, capaces de evaluar funciones con variables, cambiando cualquier dato en tiempo de ejecución. Se puede ver más información en el apéndice A.

Pensando en cómo relacionar los estados de un agente, los comportamientos que realiza y las "variables" que determinan en cada momento las propiedades del agentes, se obtienen dos modelos distintos.

3.4.1. Modelo 1.

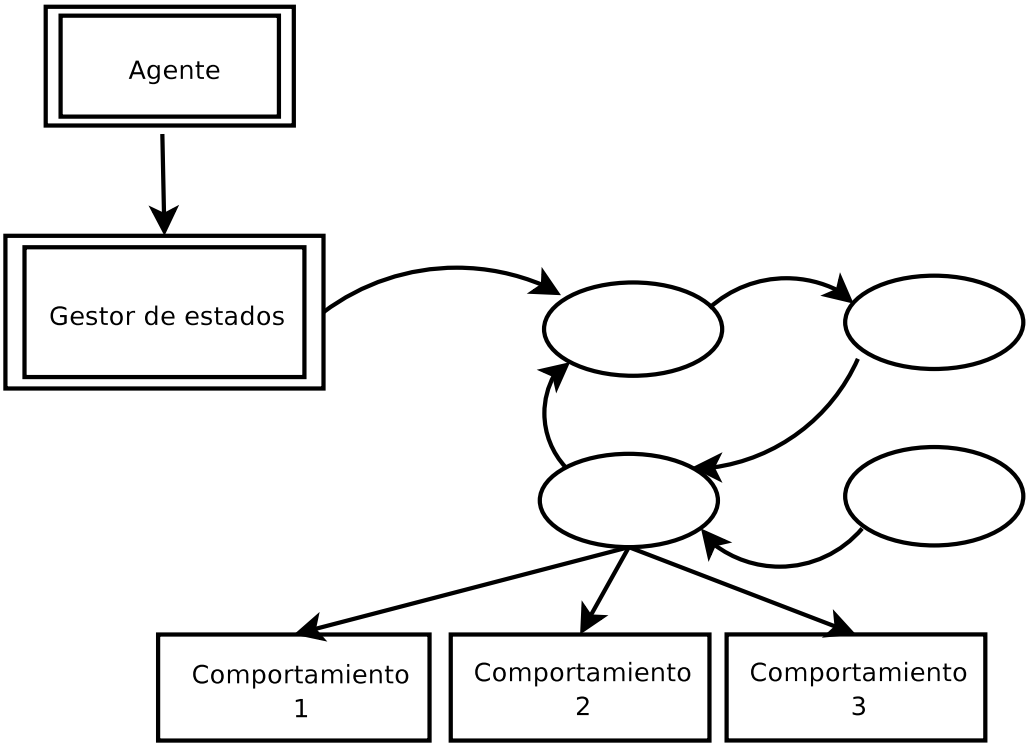

FIGURA 3.2. Modelo 1.

Estado y *comportamiento* son palabras que llevan a pensar en un autómata. Con esto en mente basamos este modelo en que las variables de los agentes determinan mediante divisiones en su rango los distintos estados del agente. Es decir, un estado se corresponde con un rango de valores de determinadas variables. Por ejemplo, la variable *energía* de un robot explorador puede determinar el paso de un estado *Mucha energía* a *Poca energía*. Dentro de cada estado, se baraja un conjunto de estrategias posibles para seleccionar.
En cada estado del autómata las variables del agente se pueden modificar de forma distinta.
En este modelo el aprendizaje nos sirve para saber, dado un estado, qué estrategia debemos seleccionar, tal y como se muestra en la figura 3.2.

3.4.2. Modelo 2.

Al igual que en el modelo anterior, se rige por un autómata. Se basa en tener tantos estados como estrategias aplicables. Las transiciones entre estados vienen determinadas por el aprendizaje. Inicialmente, desde cualquier estado se puede llegar a cualquier otro estado. De esta

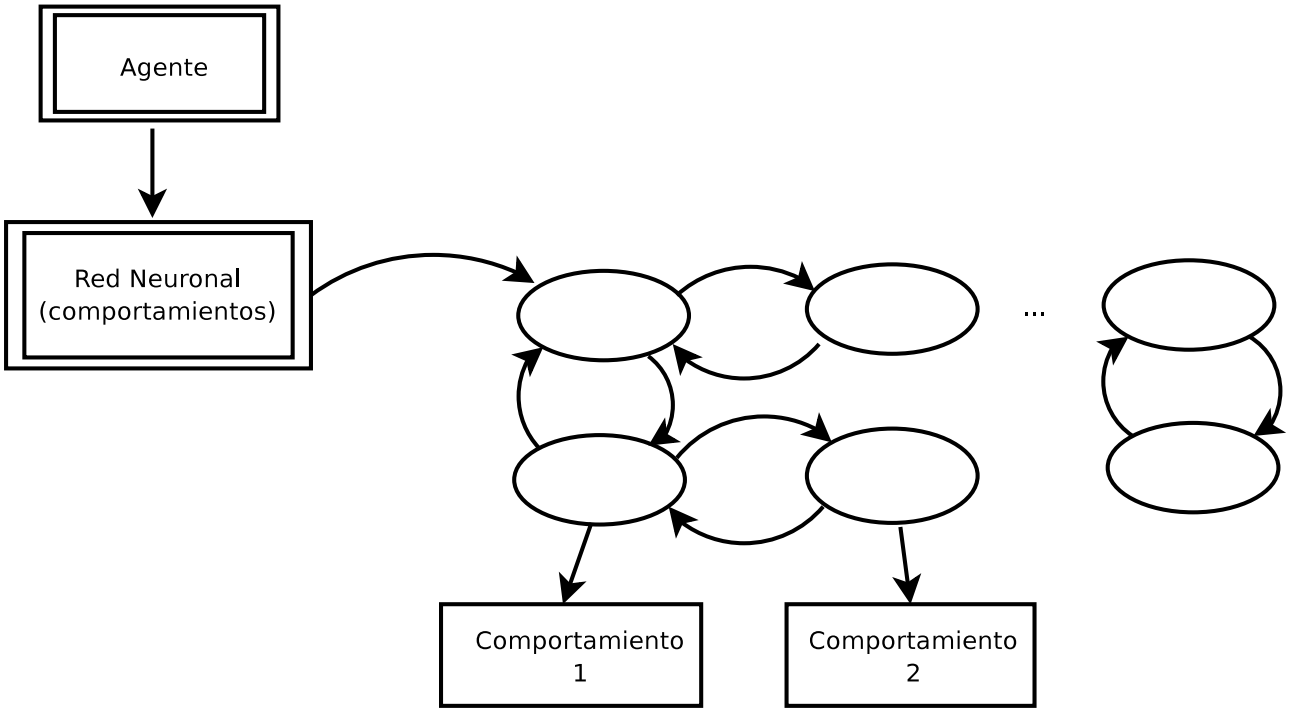

FIGURA 3.3. Modelo 2.

transición se encarga una red neuronal, de forma que tras el entrenamiento necesario, la serie de estados o estrategias escogidas por un agente nos lleven a la solución que buscamos, ya que para un estado dado y una situación, la red ha sido entrenada para cambiar al siguiente mejor estado. Su esquema se ve en la figura 3.3.

3.4.3. Conclusiones.

Ambos modelos tienen un conjunto de estrategias a aplicar dado un determinado estado, y la experiencia facilita la elección de qué estrategia escoger a continuación. Pero además, se debe tener en cuenta que se va a trabajar con varios agentes de forma simultánea. Para esto es necesaria la colaboración.

3.5. Aprendizaje

Una opción para el aprendizaje sería usar reforzamiento de las decisiones tomadas cuando se llega al objetivo deseado.

Para ello podríamos usar una tabla que contuviera los siguientes campos: Estado del agente, Acción conjunta, valoración.

Inicialmente, la primera decisión se realizaría de forma aleatoria. Es decir, las tablas de valoración de las acciones en la votación se rellenarían con valores aleatorios y así obtendríamos la primera acción conjunta. Otra posibilidad es rellenar la tabla según un conjunto de normas definidas, que hagan mas rápido el aprendizaje, pero que limitarían el conjunto de problemas a resolver, a menos que estas normas se propusieran como entrada del sistema. (Por ejemplo, considerar la proximidad de un agente para valorar sus estrategias a seguir).

Con la primera accion conjunta, se rellena el campo *estado del agente* y *acción conjunta*
En la próxima votación, se compara el estado actual con el estado anterior, y se rellena la *valoración*, mediante una función definida en la entrada del sistema
Al rellenar de nuevo las tablas de votación se vuelven a completar de forma aleatoria, o intentando guiar de algún modo las decisiones.

En el punto final, se compara el estado actual con el estado objetivo. Con la diferencia entre ambos, se ajusta la valoración de cada tabla de cada agente.
Por ejemplo, si se estima una diferencia del 30 % entre ambos estados, se sustrae 0.3 a las valoraciones de las tablas (las cuales varían entre 0 y 1).

3.6. Colaboración

Para obtener un cierto grado de colaboración, cada agente ha de conocer ciertos datos sobre el resto de los agentes, y ser capaz de decidir cuál es la acción a realiar que beneficia más al grupo.
Una de las formas de realizar acciones conjuntas beneficiosas para el grupo es hacer una votación en la que cada agente propone las acciones individuales que puede llevar a cabo, y el resto de agentes valoran cada una de estas acciones conforme al beneficio particular que reportan. La acción conjunta más votada será aquella que más beneficie al grupo en un determinado instante.

La forma en la que un agente valore las acciones individuales de los demás es mediante aprendizaje, ya que es la forma más general y menos ligada al problema.

La toma de decisiones ha de hacerse cuando se produzca un cambio de estado que pueda provocar un cambio de estrategia.

3.6.1. Votación.

- Cada agente puntúa las estrategias posibles de los demás y se construyen unas tablas donde se da valor a cada estrategia de cada agente.

	Estrategia 1	Estrategia 2	Estrategia 3
Agente 1	0.0	0.5	0.75
Agente 2	0.2	0.2	0.50
Agente 3	0.9	0.5	0.00

- Para cada agente y estrategia, se suman las celdas de todas las tablas
- Se obtiene la estrategia más votada para cada agente, y se forma el vector que define la *acción conjunta*.

3.7. Boceto de especificación

Desde la base de la situación actual de desarrollo, se puede hacer un indicio de formalización del sistema. Para ello se comienza formalizando los conjuntos de una manera abstracta, a base de fórmulas con dominios de manera que la implementación pueda llevarse a cabo de forma flexible y natural siguiendo rigurosamente las relaciones especificadas.

Se diferencia la especificación de bajo nivel, interna a un agente y por la que se definirá su comportamiento de forma local, y la especificación de alto nivel en la que se incorpora la comunicación y coordinación del sistema multi-agente completo.

El boceto de especificación sugiere una memoria interna de los agentes, en la que se almacenan los conocimientos del agente. Como veremos en el siguiente capítulo, se puede prescindir de ella y es, de hecho, la variación de este boceto hasta convertirse en la especificación final.

3.7.1. Definiciones de especificación.

Dada la ampliación sobre el concepto de agente, se propone la siguiente especificación:

- Conjunto de estados del entorno $S = S_i$ donde $S_i = (E_1, \ldots, E_n)$ y $E_i =< prop, f(I_a) >$. Aquí $prop$ es un conjunto de propiedades dependientes del problema, por ejemplo la posición en el entorno o la energía. La función $f : I_a \longrightarrow I_a$ es la encargada de indicar al elemento que interactúa lo que debe hacer. Por ejemplo, si el elemento es una pared, la función indica cómo interactúa otro elemento con él, el decir, que ningún elemento puede traspasarla.
- Conjunto de influencias que puede producir el agente en el entorno: $T_a = g(S)$ donde $g : S \longrightarrow S$ es una función que modifica el entorno según la decisión tomada para la siguiente acción del agente. Por ejemplo, tomar una decisión de movimiento implica indicar al entorno que el elemento que representa al agente ha cambiado su posición.
- Conjunto de percepciones de un agente $P_a \subseteq S_i$. Los elementos de P_a son tuplas de E, es decir, los elementos del entornoque percibe el agente a, lo que percibiría un robot por sus sensores.
- I_a es una tupla de valores que indica, de manera precisa, cuál es la situación actual del agente dentro del problema. Por ejemplo una I_a en un problema típico puede ser una terna con la posición (dos dimensiones) y la energía.
- Op_a es el conjunto de operaciones que puede realizar el agente a.
- $MemInt$ es el conjunto que contiene los elementos del entorno conocidos hasta el momento. Almacena conocimiento de bajo nivel.
- ES es el conjunto de estrategias aplicables por un agente en un determinado $Estado$ del agente.
- $Estado$ es un conjunto que define el rango de estrategias sobre el que se puede obtener ES.

- MEM es un conjunto que contiene las estrategias ES utilizadas hasta el momento y las circunstancias en que se escogió cada estrategia.
- V es el conjunto de las valoraciones sobre las estrategias de cada agente del sistema.
- EDC_a es un conjunto de valoraciones de cada estrategia que el agente a puede realizar en el $Estado$ en el que se encuentra.
- SDC_a es un conjunto de valoraciones que el agente a realiza sobre las estrategias de todos los agentes.
- RA es el conjunto de SDC menos el SDC_a del agente a que está tratando. Es el equivalente al conjunto S en este nivel.

3.7.2. Especificación de bajo nivel.

A bajo nivel se especifica el comportamiento del agente como:

Percepción: $S \longrightarrow P_a$

Es decir, las percepciones se obtendrán a partir del entorno.

Deliberación: $P_a \times ES \times MemInt \longrightarrow Op_a \times MemInt$

Lo que indica que dada una percepción, una estrategia y usando la memoria interna, se selecciona una operación, y se guarda en la memoria interna lo percibido.

Ejecución: $Op_a \times S \longrightarrow T_a$

Es decir, las operaciones aplicadas sobre el entorno producen influencias sobre el mismo.

Behaviour: $S \times ES \times MemInt \longrightarrow T_a \times MemInt$

Con esto se expresa que dado un entorno, se aplica una estrategia ayudándose de la memoria interna, obteniéndose así una influencia sobre el entorno y almacenando en memoria lo realizado.

3.7.3. Especificación de alto nivel.

A alto nivel se especifica el comportamiento del agente como:

Percepción: $V \longrightarrow EDC$

Es decir, las entradas de datos de colaboración vendrán de una votación.

Deliberación: $EDC \times MEM \times Estado \longrightarrow ES \times MEM$

Los datos obtenidos en la votación, junto con lo que se tiene almacenado en memoria y el estado actual del agente, determinan qué estrategia escoger, almacenando la elección en memoria.

Ejecución: $SDC \times RA \longrightarrow V$ La votación está compuesta por la salida de datos de colaboración de todos los agentes.

Behaviour: $SDC \times RA \times MEM \times Estado \longrightarrow ES \times MEM$

Dada una votación, un estado del agente y ayudándose de la memoria, se escoge una estrategia y se almacena la elección en memoria.

3.7.4. Especificación del comportamiento completo.

Behaviour: $SDC \times RA \times MEM \times Estado \times S \times MemInt \longrightarrow T_a \times ES \times MEM \times MemInt$

Basándose en una votación, las memorias de alto y bajo nivel, el estado del agente y el entorno, se obtiene una estrategia a seguir y se genera una influencia sobre el entorno, y además se almacena lo necesario en las memorias para su posterior uso.

3.7.5. Niveles de comunicación.

Tras la división en niveles del agente, análogamente el sistema queda dividido en niveles de actuación, a los que acompañan distintas formas de comunicación. En la figura 3.4 se muestran estos niveles.

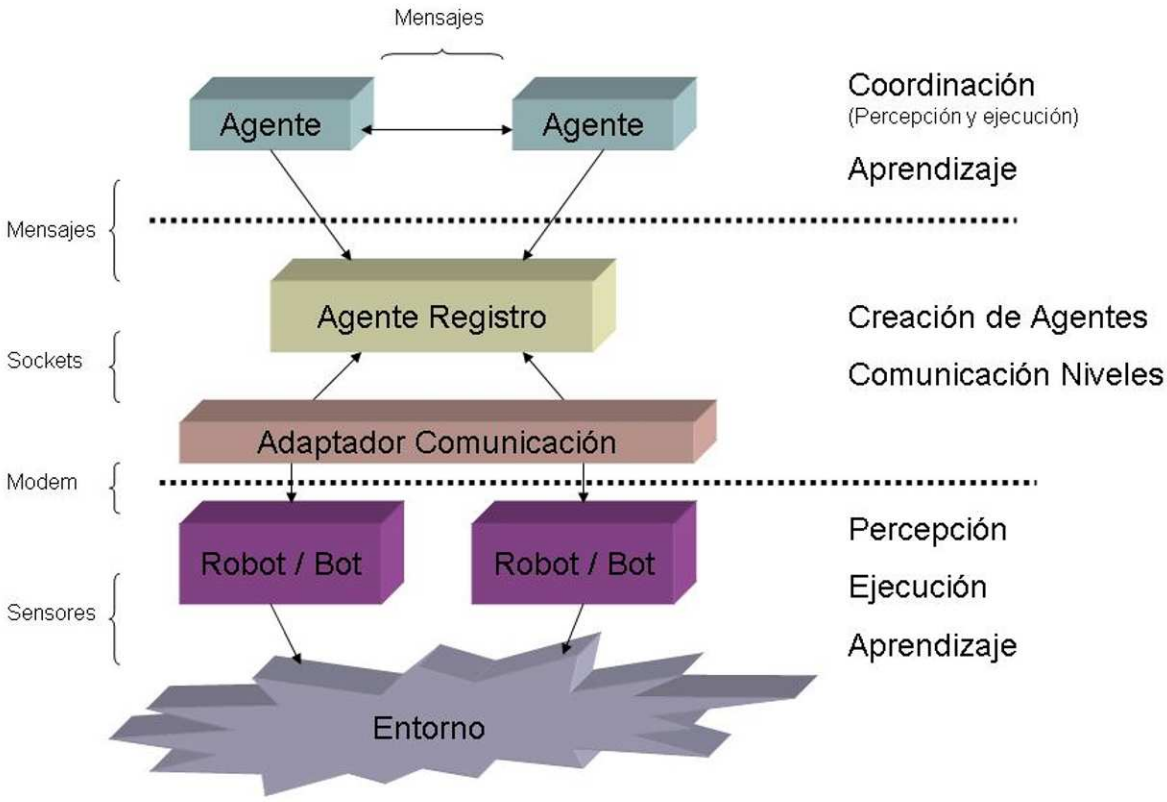

FIGURA 3.4. Niveles de actuación.

3.8. Ejemplo del nido

Aquí se indica el ejemplo típico que se ha usado durante el desarrollo del proyecto. El entorno es un laberinto desconocido para los dispositivos físicos, que serán robots. Estos robots tienen capacidad de movimiento y son capaces de detectar un “foco de comida”.

Restricciones
- Los robots buscan comida.
- Cuando la encuentran deben llevarla al nido.
- Cada montón de comida posee varias unidades de comida.
- Los robots pueden morir por inanición.

Objetivos
- Llenar el nido.
- Que sobreviva el máximo número de robots.
- Resolver el problema en el menor tiempo posible.

Estrategias
- Búsqueda
  - *Alrededor(X)*: busca sin “alejarse mucho” de una cierta zona X.
  - *Libre*: busca sin preocuparse de la distancia que recorre.
- Vuelta
  - *Directa*: Vuelve al nido directamente.
  - *Vuelta-Búsqueda*: Vuelve al nido, haciendo pequeñas búsquedas en el camino.

Movimientos
- *Búsqueda-tipo*: realiza una búsqueda en profundidad, anchura, profundidad limitada…
- *IR(X, Y)*: va directamente desde el punto X al punto Y por el camino conocido más corto.
- Otros.

Estados
- Mucha energía.
- Media energía.
- Poca energía.
- Sin energía = muerte.

Variables : Energía.

Un posible flujo de ejecución del sistema para la resolución del problema del nido:

- Inicio del sistema.
- Primera votación. ¿Qué hace cada uno?
- Los robots ejecutan sus tareas asignadas.
- Cuando sucede algún evento que da la posibilidad de que algún agente cambie de estrategia, se realiza una nueva votación. Estos eventos se dan lugar cuando el agente llega a un valor umbral de alguno de sus parámetros (ejemplo: mínimo de energía) o alguno provocado por el entorno (encontrar comida).
- El sistema para cuando el nido está lleno, o todos los agentes han muerto.

3.9. Entrada del sistema

3.9.1. Definición del problema.

El problema se definirá como un conjunto de restricciones y objetivos.

3.9.2. Estrategias.

Las estrategias que puede realizar cada agente han de ser indicadas como entrada del sistema. La forma más sencilla y a la vez más potente es usando el propio lenguaje java, facilitando una serie de funciones predefinidas, sobre movimiento, detección, etcétera.

3.9.3. Estados de un agente.

Las variables que definen el estado de un agente, las fórmulas que rigen la variación de tales variables, los estados del autómata y las estrategias posibles por estado que puede usar el agente, se deben introducir como entrada del sistema.

CAPÍTULO 4

Especifi cación

En este capítulo se muestran los resultados finales con respecto a la arquitectura desarrollada durante el proyecto.

4.1. Conceptos

Se plantea el concepto de agente clásico [**Russell04**], ampliado para participar en un sistema multiagente colaborativo. La figura 4.1 muestra esta idea.

De forma más detallada se pueden ver las partes que componen el agente con la figura 4.2.

Se consideran dos niveles tanto de percepción como de actuación. Esto se debe a que el agente ya no solo interactúa con el entorno, sino que también lo hace con el resto de agentes

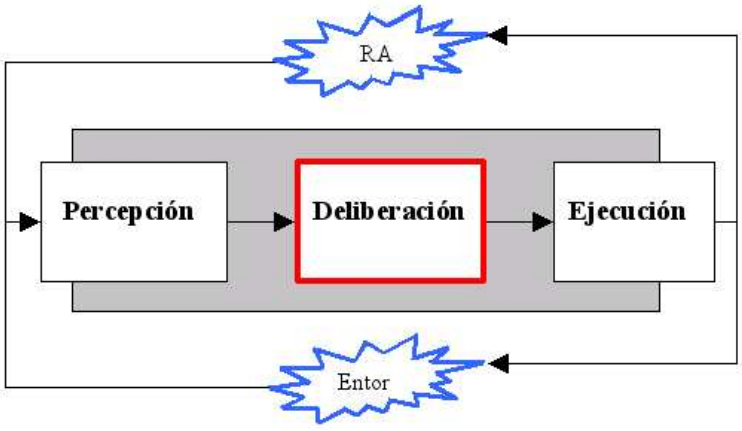

FIGURA 4.1. Sistema a alto nivel.

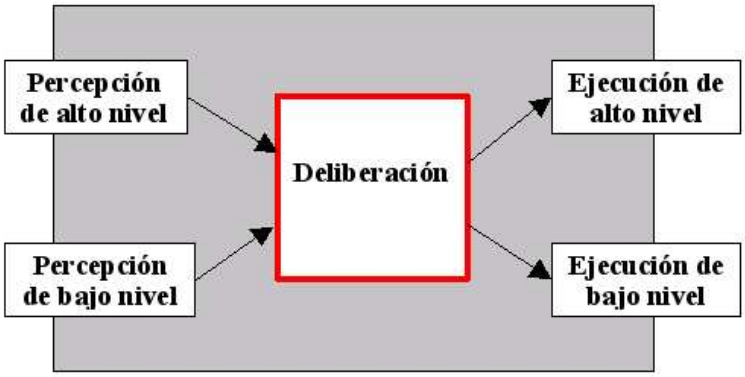

FIGURA 4.2. Sistema a bajo nivel.

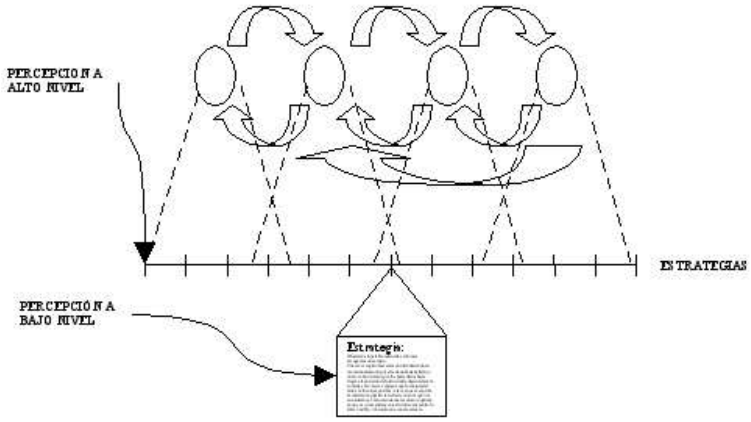

FIGURA 4.3. Niveles de percepción.

para conseguir aí colaboración.

A alto nivel, los agentes decidirán colaborativamente la estrategia que ha de seguir cada uno. Esta estrategia se usará en la ejecución a bajo nivel, tal y como se muestra en la figura 4.3.

4.2. Especificación

Dada la ampliación sobre el concepto de agente, se propone la siguiente especifiación:

- Conjunto de estados del entorno: S
- Conjunto de influencias que puede producir el agente en el entorno: T_a
- Conjunto de percepciones de un agente a: P_a

- Conjunto de estados internos del agente a: I_a
- Conjunto de operaciones que puede realizar el agente a a bajo nivel: Op_a
- Conjunto de estados del resto de los agentes: RA
- Conjunto de percepciones de un agente a: PA_a
- Conjunto de operaciones que puede realizar el agente a alto nivel: OpA_a
- Conjunto de influencias que puede producir el agente sobre el resto de agentes: TA_a
- Conjunto de objetivos: Ob

4.3. Especificaciónn de un Problema

- Definición: Problema $\longrightarrow S \times E \times RA \times Ob$
- Resolución: $S \times E \times RA \longrightarrow Ob$

4.4. Especificación de un Agente

- A bajo nivel
  - Percepción: $S \times I_a \longrightarrow P_a$ (Se percibe considerando el estado)
  - Deliberación:
    - $P_a \times I_a \longrightarrow Op_a$ (Decisión a bajo nivel)
    - $P_a \times I_a \longrightarrow I_a$ (Memorización y posible cambio de estado)
  - Ejecución: $Op_a \times S \longrightarrow T_a$ (Resultado de la operación sobre el entorno)
  - Behave: $S \times I_a \longrightarrow T_a \times I_a$
- A alto nivel
  - Percepción: $RA \times I_a \longrightarrow PA_a$ (Se percibe la situación del resto de agentes)
  - Deliberación:
    - $PA_a \times I_a \longrightarrow OpA_a$ (Decisión a alto nivel)
    - $PA_a \times I_a \longrightarrow I_a$ (Memorización y posible cambio de estado)
  - Ejecución: $OpA_a \times RA \longrightarrow TA_a$ (Influencias sobre el resto de agentes)
  - Behave: $RA \times I_a \longrightarrow TA_a \times I_a$
- Behave general
  - Behave: $S \times RA \times I_a \longrightarrow T_a \times TA_a \times I_a$

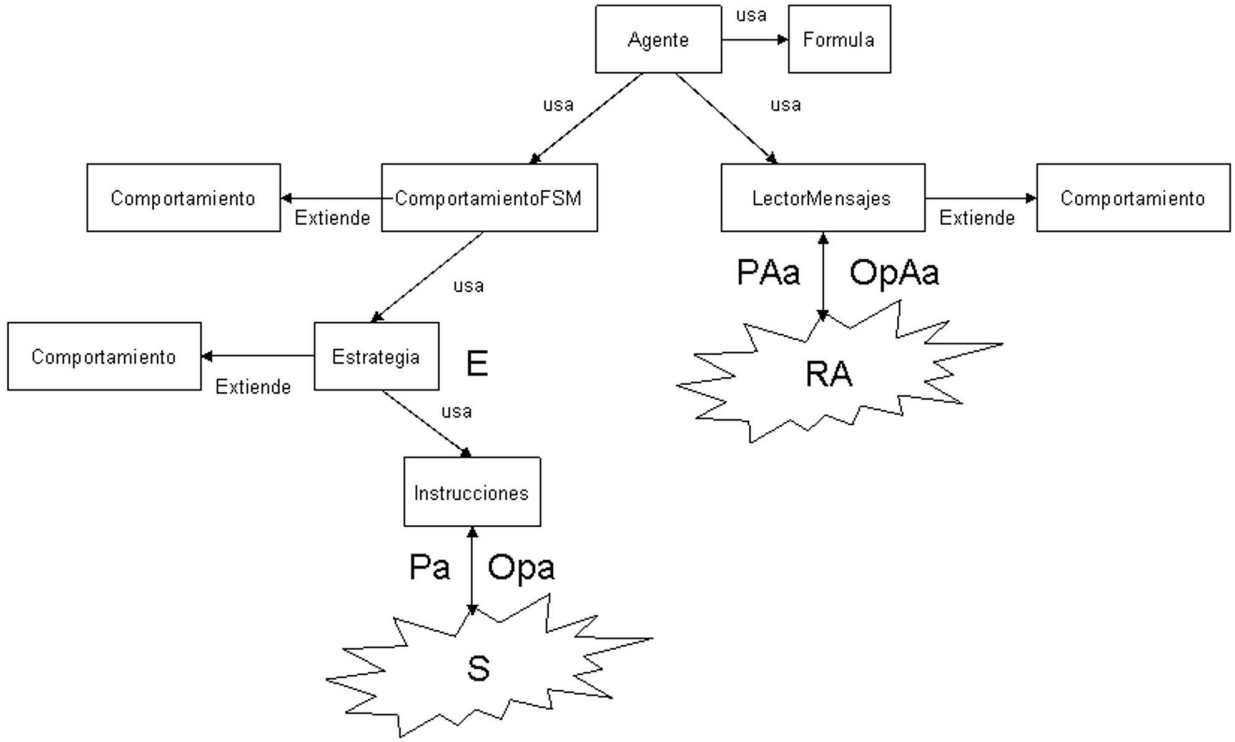

FIGURA 4.4. Esquema de la arquitectura general

I_a está compuesto por una memoria MEM y una estrategia a seguir E además del conjunto de variables.

El conjunto de variables que conforman el estado interno I_a del agente, varían siguiendo una serie de fórmulas definidas por el usuario:

$$I_a \times FORMULA \longrightarrow I_a$$

Acercando un poco esta especificación a la implementación, se obtiene la arquitectura general del agente, mostrada en la figura 4.4.

CAPÍTULO 5

Ejemplo implementado

A la hora de implementar un ejemplo para esta arquitectura de forma general se encuentran diversos problemas.
De hecho, para una arquitectura tan general, el hecho de fijar un punto de acuerdo entre la generalidad del sistema y la especificación de la entrada es un auténtico dilema. Un sistema de entrada muy simple, lo que implica poca generalidad del sistema. Una entrada muy compleja, facilitando la generalidad del sistema.
El caso extremo de simplicidad de entrada es un sistema que carece de ella, lo que implica también un mínimo de generalidad. En el otro extremo, se encuentra el diseño del problema en el propio diseño del sistema, una entrada muy compleja que proporciona la máxima generalidad.

Por todo lo anterior, se decide buscar un punto intermedio, lo que lleva a la siguiente división en módulos:

1. Interfaz gráfica de entrada. Usada para la definición del problema.
2. Entorno de simulación. Usado para simular la ejecución del sistema.
3. Sistema de agentes. Es el que proporciona la inteligencia al sistema.

5.1. Interfaz gráfica de entrada

Esta interfaz está pensada para facilitar la definición de un problema como entrada al sistema. Como característica notable de esta interfaz gráfica, cabe destacar la generación de codigo compilable que servirá como entrada al entorno de simulación y al sistema de agentes. Para ello consta de varias partes:

- Editor de mapa.

  Se usa para diseñar el entorno sobre el que trabajarán los robots. Para ello el primer paso es añadir elementos al entorno. Estos elementos serán con los que interaccione el robot en la simulación del sistema. Al crearlos, se definen una serie de parámetros que especificarán su comportamiento. Por ejemplo se puede indicar que:

  - Es un sumidero: es decir, cuando un robot interactua con el, modifica negativamente algún parámetro.
  - Es un foco: en su interacción proporciona un aumento de cierto parámetro del robot.
  - Es opaco: indica que un elemento dinámico no puede pasar a través de este elemento.
  - Es estático: indica que no se moverá en el entorno, lo que hará que se trate de una forma mas sencilla.
  - Es token: es un elemento que el robot debe recoger y llevar a algún sitio.
  - Es CheckPoint: indica que el robot debe pasar por este sitio.

  Una vez definidos los elementos, se puede ir creando el entorno simplemente escogiendo uno de estos elementos y colocándolo en el mapa. La figura 5.1 muestra una captura de pantalla del interfaz.

- Diseño de los estados:

  Aquí se pueden crear estados que serán usados posteriormente como parte de los agentes. A estos estados se les pueden asociar parámetros, un evento o condición que determina los límites de los parámetros para los que el agente debe permanecer en ese

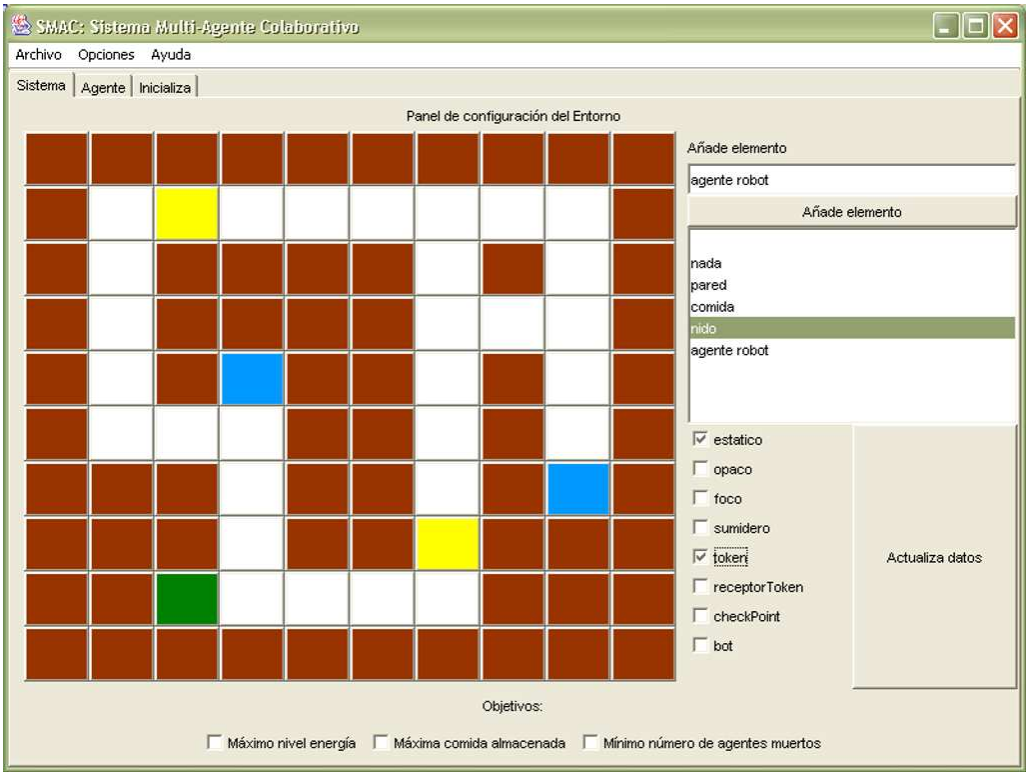

FIGURA 5.1. Edición del mapa del entorno.

estado. Además se asocian las funciones por las que los parámetros de un estado irán variando. También podemos asociar valores iniciales a los parámetros definidos para los elementos creados en el editor de mapa. La figura 5.2 muestra una captura de pantalla de esta pestaña.

- Valores iniciales:

  Desde esta parte del interfaz, se dan los valores iniciales a los elementos creados, como se puede ver en la figura 5.3.

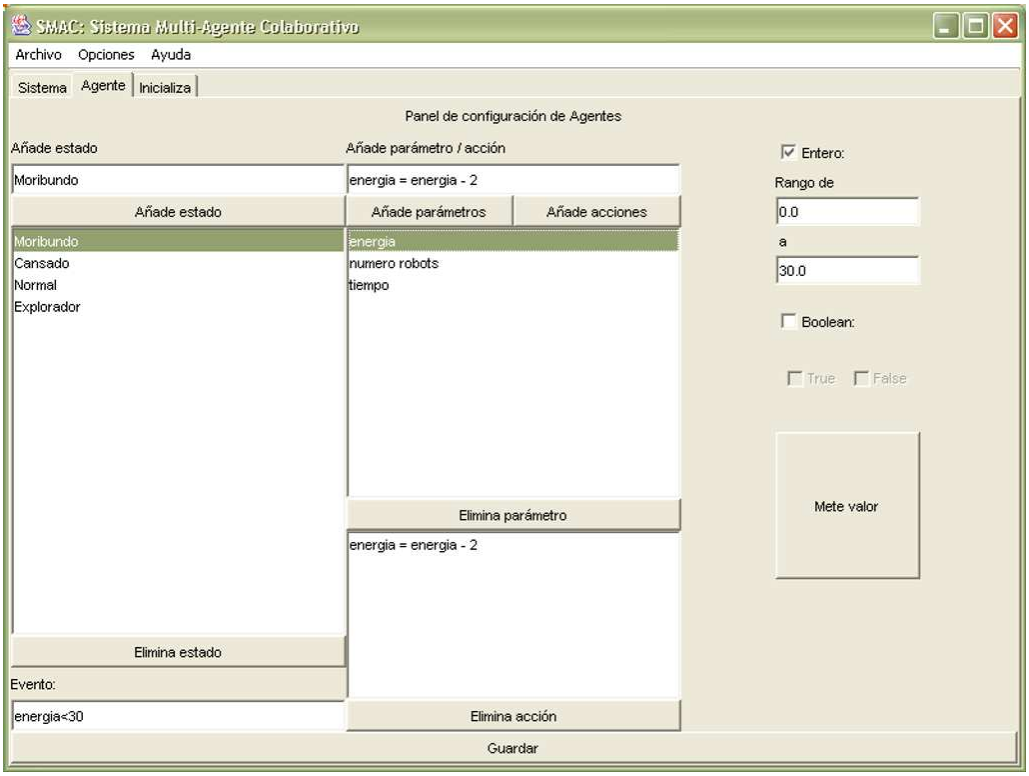

FIGURA 5.2. Diseño de los estados.

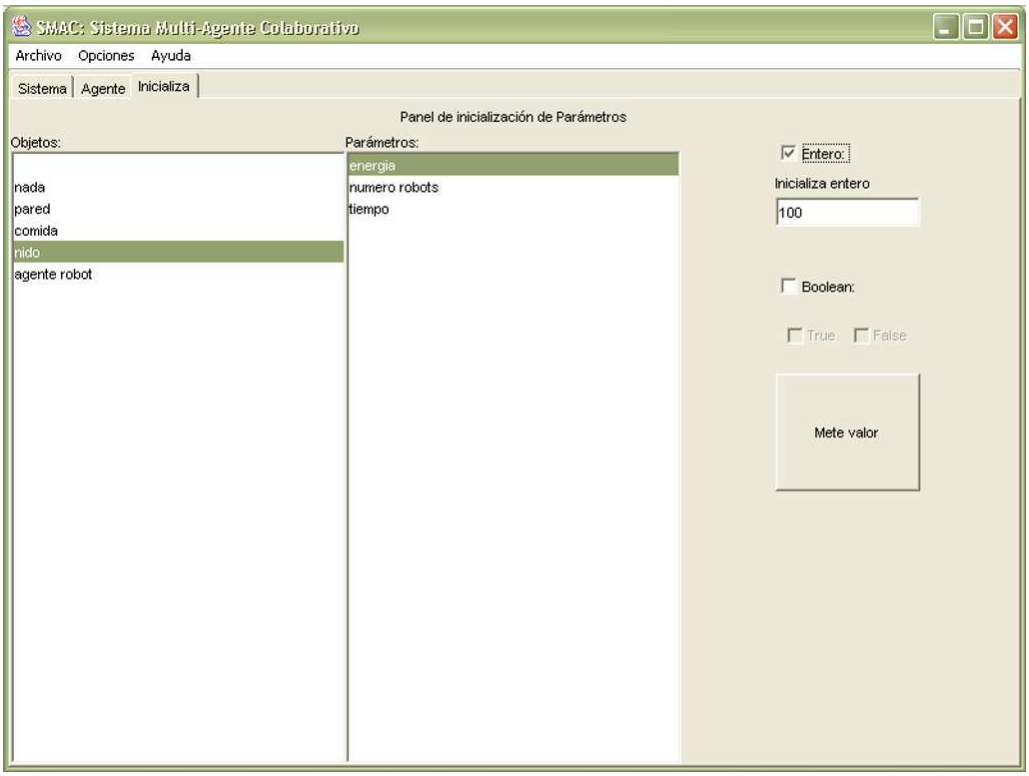

FIGURA 5.3. Valores iniciales de los elementos.

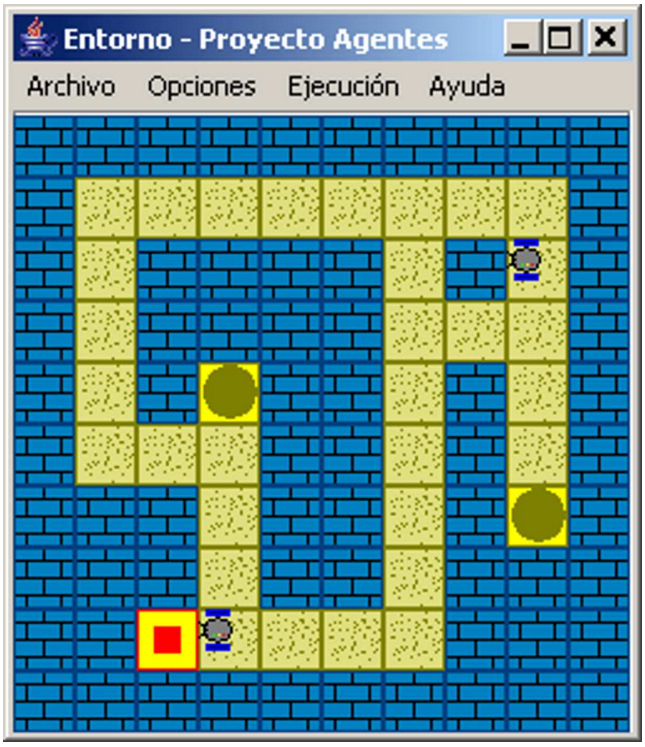

FIGURA 5.4. Captura de la aplicación en ejecución.

5.2. Entorno de simulación

Éste módulo se ha implementado para llevar a cabo la simulación del sistema. Consta de una interfaz gráfica, para dar al usuario capacidad de controlar la simulación, y a su vez de representar gráficamente el estado del problema, mostrando el entorno y la posición de los robots. Esta interfaz se muestra en la figura 5.4.

Este módulo es el encargado de dirigir todo el proceso, ya que mantiene las estructuras de datos necesarias para mantener el entorno, y además manejar los objetos encargados de la interconexión de la simulación con el sistema de agentes. Se puede ver su diagrama de clases en las figuras 5.5 y 5.6.

Para crear esta interconexión, se ha usado un paquete (ServidorComunicaciones) que se encarga de crear un servidor, y escuchar peticiones por parte de los agentes. También contiene objetos manejadores de dichas conexiones, para solucionar los problemas que pueden surgir al manejar *Sockets*, tales como la lectura bloqueante.

Una de las tareas del entorno, es lanzar un contenedor de sistemas multiagente, y el servidor de comunicaciones para recibir conexiones. En el sistema multiagente se carga un agente por cada

robot, que inicia una conexión contra el servidor de comunicaciones. Este servidor proporcionará un manejador de comunicaciones al robot correspondiente.
En este entorno se hace uso de los elementos generados en la interfaz gráfica de entrada. Para mantener estos elementos, guarda una matriz de elementos estáticos, y una lista de elementos dinámicos. La separación de estos tipos de elemento se debe a que se tratarán de una forma distinta.

Por ejemplo, a la hora de ejecutar la simulación, se hace que los elementos dinámicos actúen. En este caso concreto, se ha usado solamente un elemento dinámico, que representa un robot. El robot tiene un conjunto de sensores y un conjunto de actuadores. La ejecución del robot puede verse como:

- Percepción del entorno.
- Comunicación de lo percibido.
- Recepción de las acciones a tomar.
- Actuación.

La percepción del entorno, en el caso de la simulación se hace directamente preguntando al entorno que es lo que tiene al alcance de sus sensores. La comunicación de lo percibido al sistema de agentes, así como la recepción de las acciones a tomar por parte del mismo, viene facilitada por la clase ManejadorComunicacion, que mantiene una conexión via *Socket* con el agente correspondiente.

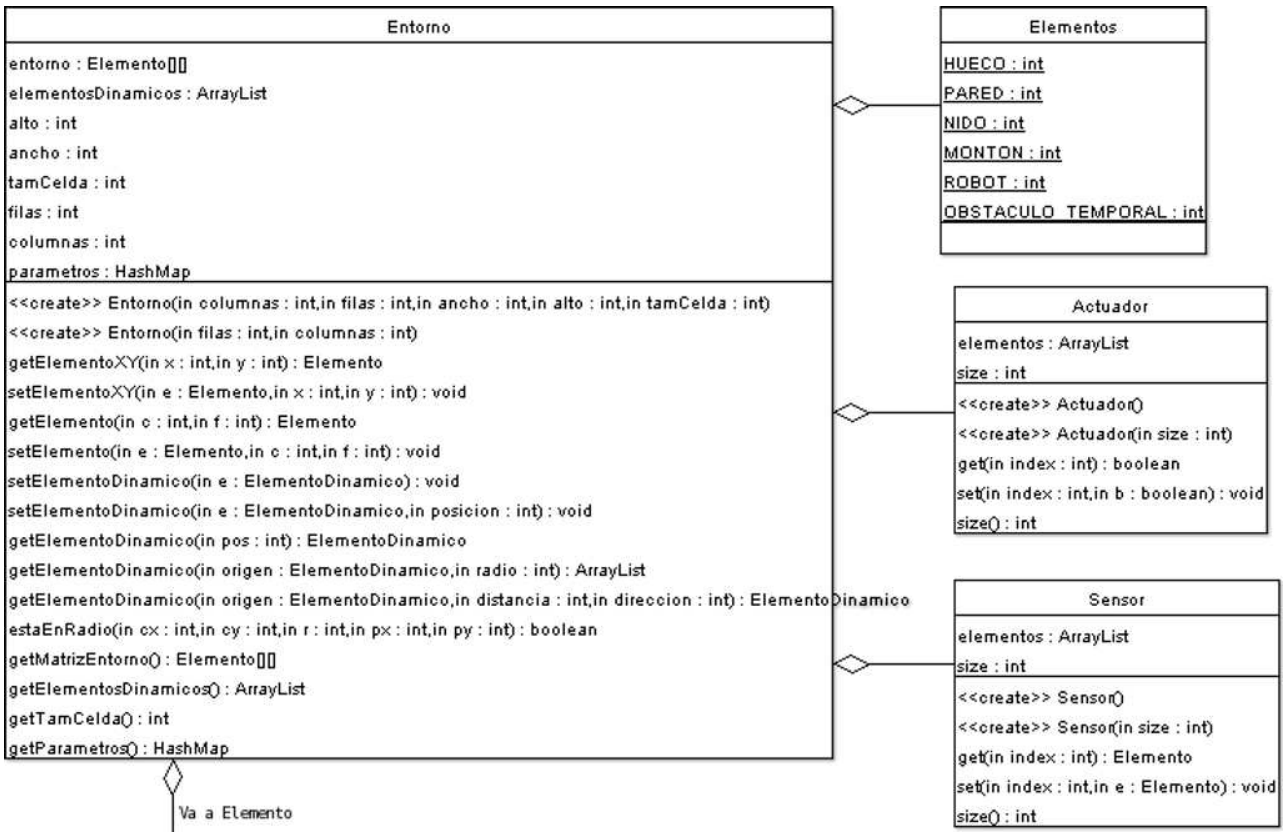

FIGURA 5.5. Primera parte del diagrama de clases de Entorno

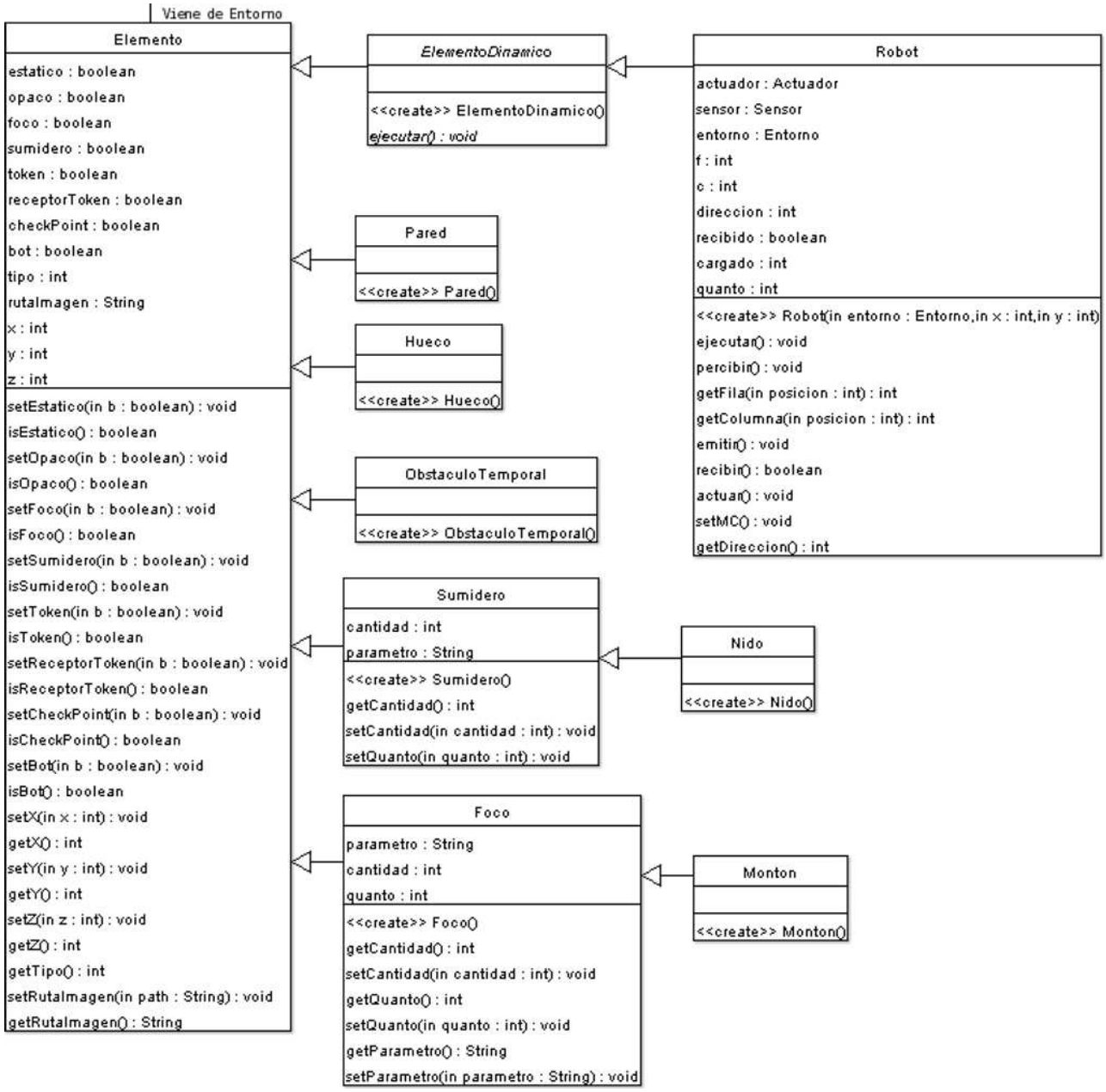

FIGURA 5.6. Segunda parte del diagrama de clases de Entorno

5.3. Sistema de agentes

Este sistema está implementado sobre la plataforma JADE. El sistema básico se compone de una serie de clases que se usarán para generar el código final del sistema.
Por ejemplo, las clases básicas para generar un agente, un estado o una estrategia están implementadas, listas para usarse como origen de herencia. A partir de la interfaz gráfica de entrada, ya que guarda los datos del problema, se pueden generar el resto de ficheros para hacer el sistema completamente funcional.
Estos ficheros son los que actúan sobre parámetros concretos del sistema, como son los comportamientos, las estrategias, y el autómata que rige la evolución de los parámetros del agente.

La arquitectura que implementa este sistema de agentes puede verse en la figura 5.7.

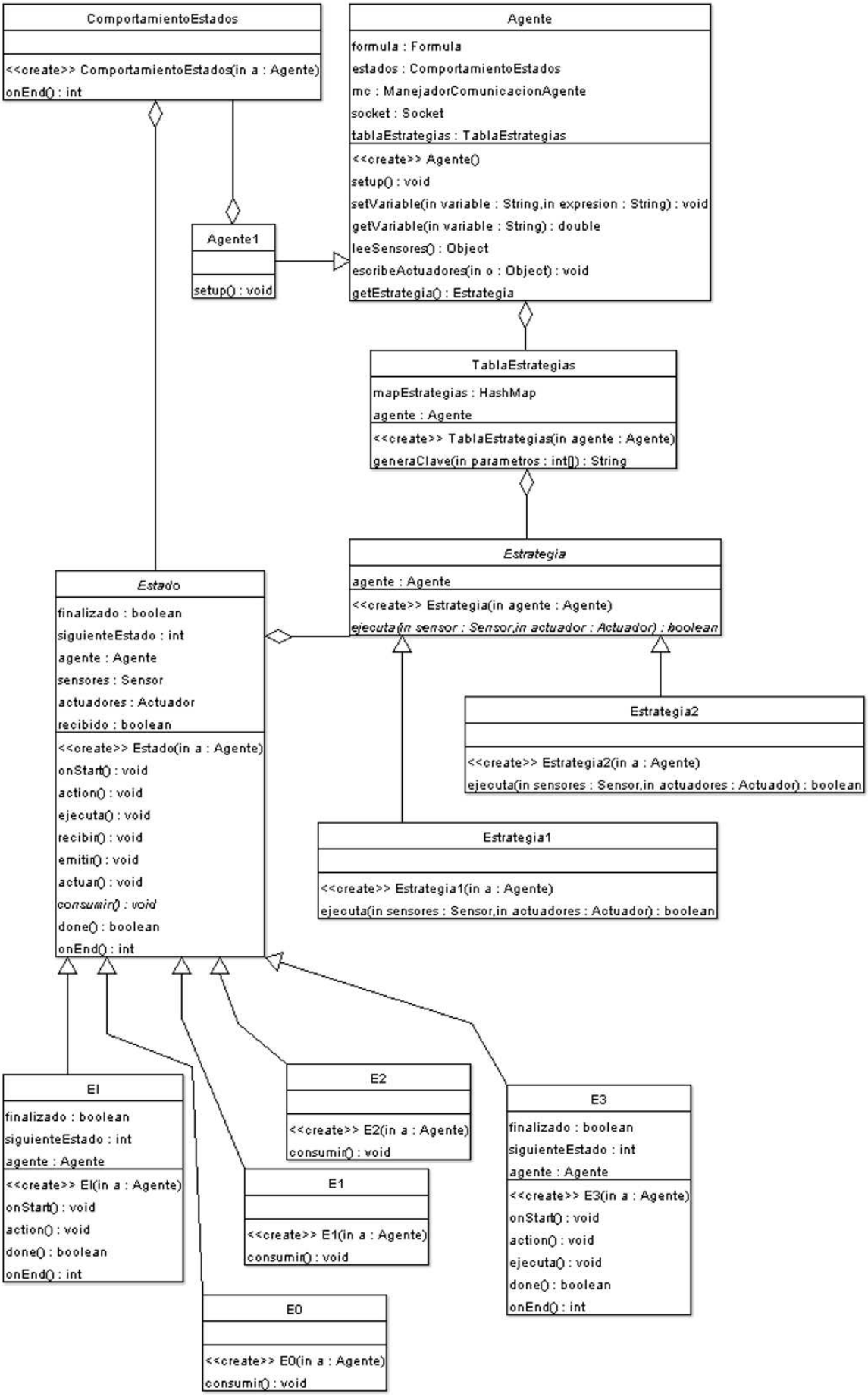

FIGURA 5.7. Diagrama de clases de la arquitectura implementada

Para llevar a cabo el diseño de las partes concretas de cada problema hay que generar las siguientes clases:

Se genera un agente por cada tipo de robot. En este agente lo que hay que implementar es simplemente el autómata, que se hace de forma muy simple a través del objeto estados, de la clase ComportamientoEstdos, que extiende a la clase FSMBehaviour perteneciente al api proporcionado de JADE..

Además del agente, hay que generar cada uno de los comportamientos que irán asociado a cada estado del autómata. Para cada uno de estos estados solo hay que implementar el método ’consumir( )’, que lleva acabo la modificación de los parámetros del agente.

Una parte esencial, y ajustada al tipo de problema son las estrategias. Para implementar una estrategia hay que extender la clase Estrategia, e implementar el método ’ejecuta( )’, donde se puede hacer uso de los atributos ’sensores’ y ’actuadores’ para poder obtener información del robot y enviar acciones a realizar.

Además de las estrategias, hay que modificar la clase TablaEstrategias, que contiene una tabla que relaciona el estado del agente con la estrategia a seguir. La clave de esta tabla se hace a partir de los parámetros definidos para el agente, y de la función de transición del autómata.

Esta tabla está pensada para ser modificada durante el aprendizaje. En este caso concreto se ha pensado en un aprendizaje basado en algoritmos evolutivos. Los pasos para este aprendizaje serían:

- Se genera una población inicial, compuesta por conjuntos de agentes, cada uno con su tabla de estrategias.
- Se evalúa el sistema, ejecutándolo y calculando la distancia de los parámetros del sistema que intervienen en los objetivos, y se seleccionan aquellas poblaciones cuyo resultado obtenido se aproxime más.

- Aleatoriamente se genera un punto de cruce en una determinada clave de la tabla, y se intercambian los datos a uno y otro lado entre los agentes de las poblaciones.
- Se itera hasta conseguir el nivel de aprendizaje deseado.

Bibliografía

[Bigus01] Joseph P. Bigus and Jennifer Bigus. *Constructing intelligent agents usign JAVA*. Wiley. New York. 2001.

[Cortese02] E. Cortese, F. Quarta and G. Vitaglione. *Scalability and Performance of JADE Message Transport System*. 2002. http://jade.tilab.com/papers/Final-ScalPerfMessJADE.pdf.

[Mestre04] J.L. Mestre, I. Luño and I. Morales. *Servidor de Agentes*. Proyecto de Sistemas Informáticos del curso 2003/2004. 2004. Facultad de Informática, Universidad Complutense de Madrid.

[Russell04] S. Russell and P.Norvig. *Inteligencia Artificial. Un enfoque moderno. Segunda edición*. Pearson / Prentice Hall. Madrid. 2004.

[Subrahmanian00] V. S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan and Robert Ross. *Heterogeneous agent systems*. MIT Press, cop. Cambridge, Massachusetts. 2000.

[Vlassis03] N. Vlassis. *A Concise Introduction to Multiagent Systems and Distributed AI*. Informatics Institute, University of Amsterdam. sep 2003. http://www.science.uva.nl/˜vlassis/cimasdai.

APÉNDICE A

Fórmulas

Como ayuda para la implementación de los estados de un agente, se ha implementado un sistema capaz de almacenar funciones y evaluarlas.
Este sistema, llamado '*Formula*', posee características similares a un traductor de lenguaje. Esto se debe a que si queremos manejar fórmulas desde el codigo fuente sin que se evaluen, han de guardarse en cierto formato, que pueda ser posteriormente evaluado.
El formato elegido para representar las fórmulas es una cadena de caracteres.
El módulo '*Formula*' posee un analizador léxico que transforma estas cadenas en objetos manejables para un analizador sintáctico también incluido en el propio módulo. Básicamente este analizador léxico detecta si lo que tiene en la entrada es un numero real, un número entero o una secuencia de letras, que representaría una variable. El analizador sintáctico se ha hecho siguiendo la siguiente gramática independiente de contexto:

$E \longrightarrow TR$
$R \longrightarrow +TR$
$|-TR$
$|\lambda$
$T \longrightarrow CZ$
$Z \longrightarrow *CZ$
$|/CZ$
$|\lambda$
$C \longrightarrow FK$
$K \longrightarrow \hat{}FK$

$|\lambda$

$F \longrightarrow -F$

$|id$

$|fun(E)$

$|(E)$

$|N$

Esta gramática es no ambigua y está factorizada, para que el analizador sintáctico pueda ser predictivo, y así ser eficiente.
A la vez que el analizador sintáctico comprueba la corrección de una fórmula, va generando un árbol de expresiones, que servirá en un futuro para evaluar dicha fórmula. De esta forma el proceso de análisis solo se hace una vez, y la evaluación se simplifica al mantener este árbol de expresiones explícitamente.

Como caso especial podemos observar la producción $F \longrightarrow fun(E)$, que viene a representar una serie de funciones unarias implementadas en java. Mas concretamente se han implementado todas las funciones matemáticas que proporciona el api Math de java, que contienen un solo operador.

Las funciones que permite realizar un objeto '*Formula*' son:

- Asignar una expresión a una variable. Es decir, crear una ecuación done el lado izquierdo está compuesto por una sola variable, y el lado derecho por cualquier tipo de expresión aritmética y funcional que permita la gramática anterior.
- Obtener el valor de una variable. Al obtener el valor, se evalúa el árbol de expresiones que hay asociado a esta variable. Como parte de este árbol puede haber otras variables, que llevarían asociadas sus propios árboles de expresiones.
- Obtener los nombres de todas las variables de la fórmula. Se pueden obtener todos los nombres de variables de los lados izquierdo de las fórmulas introducidas. Es un método

útil para una vez dado un objeto ’*Formula*’, poder obtener el valor de sus expresiones sin conocer el nombre de sus variables.

En el ejemplo implementado, cada agente hace uso de su propia ’*Formula*’ en la que almacena las ecuaciones asociadas a los parámetros que definen el agente. En el caso concreto del ejemplo, los parámetros que se utilizan son ’e’, que representa la cantidad de energía que tiene el robot, y ’cargado’, que indica si el robot ha recogido comida o no.

Como ejemplo del último método disponible en la interfaz ’*Formula*’, obtener el nombre de las variables, puede verse la forma de generar claves en el sistema para acceder a la tabla de estrategias situada en TablaEstrategias. El agente obtiene los nombres de las variables de sus ecuaciones, para después obtener los valores y enviarlos al objeto que instancia la tabla de estrategias. Este objeto generará una clave a partir de estos valores, y consultará la tabla con ellos, con lo que podrá devolver la estrategia correspondiente a los valores de la fórmula.

Con esto conseguimos asociar una estrategia a cada estado del agente, siempre y cuando la función de generación de claves considere los intervalos oportunos de los parámetros del agente, que suelen corresponder con los intervalos que definen la estancia o no del autómata del agente en un determinado estado.

En la figura A.1 se muestra el diagrama de clases de ’*Formula*’.

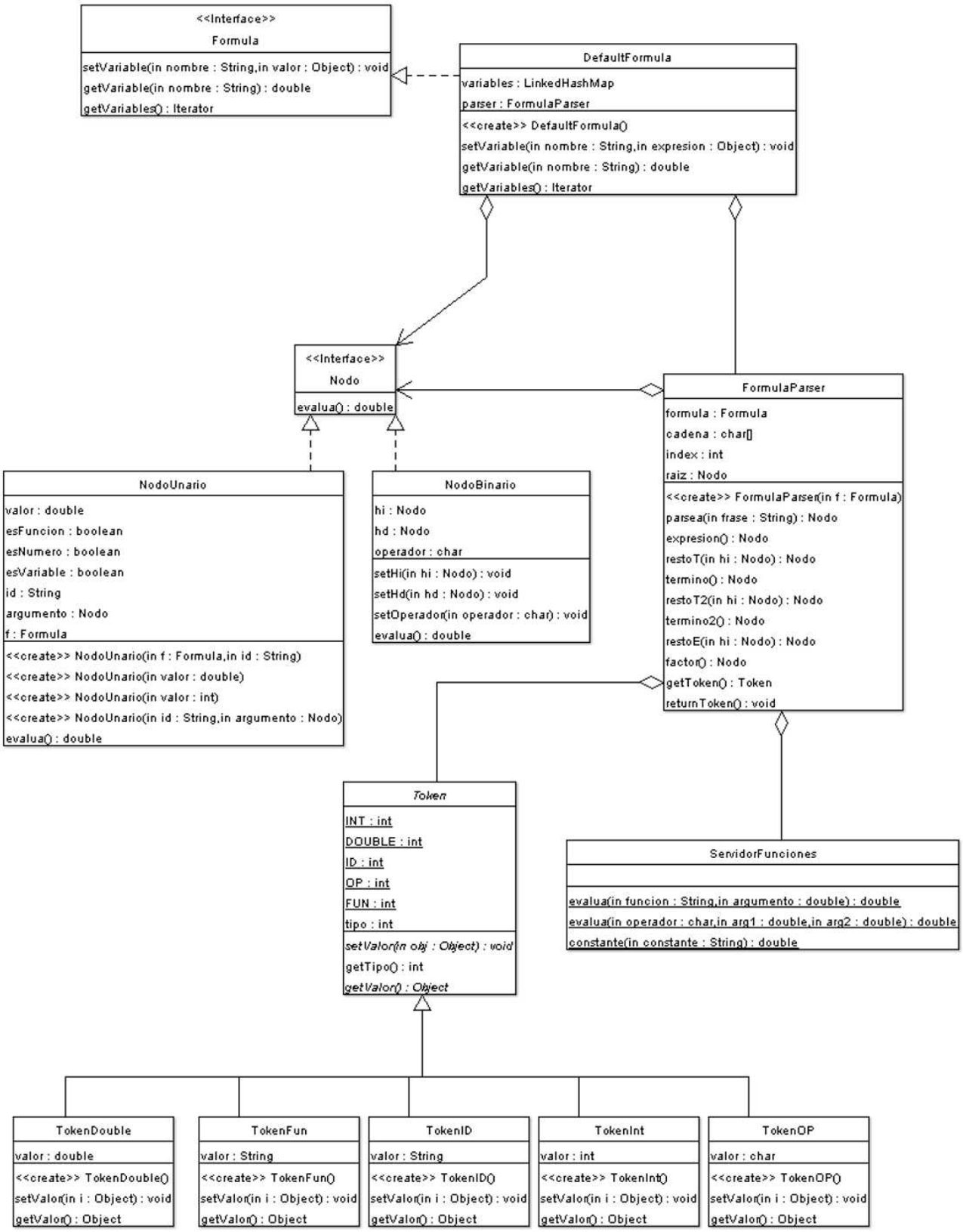

FIGURA A.1. Diagrama de clases de *Formula*

APÉNDICE B

Robots

A comienzos del curso el ámbito del proyecto incluía la incorporación de una parte física, que con el desarrollo acabó por ser eliminada debido al escaso tiempo disponible y a la inmensa complejidad que introducía esta parte física ya de por sí.

La integración con la parte lógica en el estado de la arquitectura final es inmediata, sin embargo no está tratada la parte de sensores, los cuales por su condición introducen errores y problemas derivados que habría que observar detenidamente.

Para manejar esto de una manera coherente con el objetivo del proyecto, se buscaron soluciones ya fabricadas para intentar disminuir en lo posible la diferencia de los sensores teóricos implementados en la simulación, con los reales con sus condiciones, y evitar así que debido a nuestra falta de experiencia en la construcción de estos elementos se aumentase de sobremanera la complejidad del proyecto.

Se realizó un estudio de diversos robots comerciales que servían bastante bien para el objetivo. Se presentan a continuación:

- **Rug Warrior Pro**

URL : http://www.robotbooks.com/rug_warrior.htm
Ensamblaje : Si, soldando
Comunicación RF : Por nuestra cuenta. No asegurado
E/S : LCD display, detector de colisiones, det. Infrarrojo, sensores de luz,
Conexión : serie

Precio : 585$

Notas : For Advanced Students. With powerful processor, 2-line A/N LCD display, 32K RAM, RS232 serial port, collision detector, infrared detector, light sensors, microphone, piezoelectric buzzer, motor driver chip, dual shaft encoders, user-controllable LEDs and more. Includes processing, memory/sensor circuitry, motors, chassis, custom body parts and PC board. Requires assembly and soldering.

- **Talrik Jr. Pro**

URL : http://www.mekatronix.com/detailed/tjpro.htm

Ensamblaje : opcional

Comunicación RF : con la expansión “argos” y expansión RF

E/S :

  - Two forward-looking IR emitters, emitting at a wavelength of 940nm
  - One backward-looking IR emitters, emitting at a wavelength of 940nm
  - Two forward-looking analog IR detectors. These sensors produce analog channel readings from about 88 to 128 out of a possible 256. The number 256 corresponds to five volts.
  - Three front bumper momentary tactile switches, each switch closure separately identifiable.
  - One back bumper momentary tactile switch.
  - User expandable sensors.

Conexión : RS232

Precio : 190$ montaje total.

- **Lego Mindsorms Robotics Invention System 2.0**

URL : http://www.robotstore.com/legomindstorms.asp?afid=legosq

Ensamblaje : Si, sin soldar.

Comunicación RF : No. Hay algún proyecto pero modificando IR, hardware y software (incluido legOS del RCX).

E/S : 2 sensores tacto, 1 sensor de luz, un transmisor infrarrojo.

Conexión : USB

Precio : 169.99$

- **Boe-Bot**

URL : http://www.parallax.com/html_pages/robotics/boebot/boebot.asp

Ensamblaje : Si, sin soldar.

Comunicación RF : Módulo receptor a 418MHz con 5 funciones(no emisor). Transceiver (emisor/receptor) Necesario si queremos sensores y efectores remotos.

E/S : LEDs, speaker, pushbutton, photoresistors, resistors and capacitors, infrared LEDs and receivers.

Conexión : serie/usb

Precio : 229$

Nota : Según la página, es muy popular (90.000 unidades vendidas). Hay que elegir el chip Stamp que se quiera (I, II, u otras 2 expansiones)

- **Circular Robot - IdMind**

URL : http://www.idmind.pt/English/Produtos/RobotCircular/RobotCircular.htm

Ensamblaje : Si, hay que soldar.

Comunicación RF : Por nuestra cuenta. No asegurado.

E/S :

  - Controller board (populated with a Microchip PIC-16F76)
  - Digital interface board (11 I/O ports + 2 PWM)
  - Analog interface board (4 analog ports + 1 I/O port)
  - Serial programming board
  - Base with interface for sensors (7 analog ports)
  - 2 Servomotors
  - 7 Infra-red emitters/receivers
  - 2 micro-switches

Conexión : serie

Precio : 244,65 Euros

Nota : montaje 85 euros por kit. Es portugués.